嵌入式系统和 ucOS 的一些概念

● 软实时系统和硬实时系统。在软实时系统中系统的宗旨是使各个任务运行得越快越好，并不要求限定某一任务必须在多长时间内完成。在硬实时系统中，各任务不仅要执行无误而且要做到准时。大多数实时系统是二者的结合。

● 不复杂的小系统一般可称为前后台系统或超循环系统(Super-Loops)。应用程序是一个无限的循环，循环中调用相应的函数完成相应的操作，这部分可以看成后台行为(background)。中断服务程序处理异步事件，这部分可以看成前台行为（foreground）。后台也可以叫做任务级。前台也叫中断级。时间相关性很强的关键操作(Critical operation)一定是靠中断服务来保证的。因为中断服务提供的信息一直要等到后台程序走到该处理这个信息这一步时才能得到处理，这种系统在处理信息的及时性上，比实际可以做到的要差。这个指标称作任务级响应时间。最坏情况下的任务级响应时间取决于整个循环的执行时间。因为循环的执行时间不是常数，程序经过某一特定部分的准确时间也是不能确定的。进而，如果程序修改了，循环的时序也会受到影响

● 共享资源

可以被一个以上任务使用的资源叫做共享资源。为了防止数据被破坏，每个任务在与共享资源打交道时，必须独占该资源。这叫做互斥（mutual exclusion）。多任务运行的实现实际上是靠 CPU(中央处理单元)在许多任务之间转换、调度。CPU 只有一个，轮番服务于一系列任务中的某一个。多任务运行很像前后台系统，但后台任务有多个。多任务运行使 CPU 的利用率得到最大的发挥，并使应用程序模块化。

● 任务

一个任务，也称作一个线程，是一个简单的程序，该程序可以认为 CPU 完全只属该程序自己。实时应用程序的设计过程，包括如何把问题分割成多个任务，每个任务都是整个应用的某一部分，每个任务被赋予一定的优先级，有它自己的一套 CPU 寄存器和自己的栈空间。

每个任务都是一个无限的循环。每个任务都处在以下 5 种状态之一的状态下，这 5 种状态是休眠态，就绪态、运行态、挂起态(等待某一事件发生)和被中断态。

休眠态相当于该任务驻留在内存中，但并不被多任务内核所调度。

就绪意味着该任务已经准备好，可以运行了，但由于该任务的优先级比正在运行的任务的优先级低，还暂时不能运行。

运行态的任务是指该任务掌握了 CPU 的控制权，正在运行中。

挂起状态也可以叫做等待事件态 WAITING，指该任务在等待，等待某一事件的发生，（例如等待某外设的 I/O 操作，等待某共享资源由暂不能使用变成能使用状态，等待定时脉冲的到来或等待超时信号的到来以结束目前的等待，等等）。

发生中断时，CPU 提供相应的中断服务，原来正在运行的任务暂不能运行，就进入了被中断状态。

● 任务切换(Context Switch or Task Switch)

当多任务内核决定运行另外的任务时，它保存正在运行任务的当前状态（Context），即 CPU 寄存器中的全部内容。这些内容保存在任务的当前状况保存区（Task’s Context Storage area），也就是任务自己的栈区之中。入栈工作完成以后，就是把下一个将要运行的任务的当前状况从该任务的栈中重新装入 CPU 的寄存器，并开始下一个任务的运行。这个过程叫做任务切换。任务切换过程增加了应用程序的额外负荷。CPU 的内部寄存器越多，额外负荷就越重。做任务切换所需要的时间取决于 CPU 有多少寄存器要入栈。实时内核的性能不应该以每秒钟能做多少次任务切换来评价

● 内核（Kernel）

多任务系统中，内核负责管理各个任务，或者说为每个任务分配 CPU 时间，并且负责任务之间的通讯。内核提供的基本服务是任务切换。之所以使用实时内核可以大大简化应用系统的设计，是因为实时内核允许将应用分成若干个

任务，由实时内核来管理它们。内核本身也增加了应用程序的额外负荷，代码空间增加 ROM 的用量，内核本身的数据结构增加了 RAM 的用量。但更主要的是，每个任务要有自己的栈空间，这一块吃起内存来是相当厉害的。内核本身对 CPU 的占用时间一般在 2 到 5 个百分点之间。

- 调度（Scheduler）

调度（Scheduler）,英文还有一词叫 dispatcher，也是调度的意思。这是内核的主要职责之一，就是要决定该轮到哪个任务运行了。多数实时内核是基于优先级调度法的。每个任务根据其重要程度的不同被赋予一定的优先级。基于优先级的调度法指，CPU 总是让处在就绪态的优先级最高的任务先运行。然而，究竟何时让高优先级任务掌握 CPU 的使用权，有两种不同的情况，这要看用的是什么类型的内核，是不可剥夺型的还是可剥夺型内核。

不可剥夺型内核要求每个任务自我放弃 CPU 的所有权。不可剥夺型调度法也称作合作型多任务，各个任务彼此合作共享一个 CPU。异步事件还是由中断服务来处理。中断服务可以使一个高优先级的任务由挂起状态变为就绪状态。但中断服务以后控制权还是回到原来被中断了的那个任务，直到该任务主动放弃 CPU 的使用权时，那个高优先级的任务才能获得 CPU 的使用权。

不可剥夺型内核的一个优点是响应中断快。在讨论中断响应时会进一步涉及这个问题。在任务级，不可剥夺型内核允许使用不可重入函数。函数的可重入性以后会讨论。每个任务都可以调用非可重入性函数，而不必担心其它任务可能正在使用该函数，从而造成数据的破坏。因为每个任务要运行到完成时才释放 CPU 的控制权。当然该不可重入型函数本身不得有放弃 CPU 控制权的企图。

使用不可剥夺型内核时，任务级响应时间比前后台系统快得多。此时的任务级响应时间取决于最长的任务执行时间。不可剥夺型内核的另一个优点是，几乎不需要使用信号量保护共享数据。不可剥夺型内核的最大缺陷在于其响应时间。高优先级的任务已经进入就绪态，但还不能运行，要等，也许要等很长时间，直到当前运行着的任务释放 CPU。与前后系统一样，不可剥夺型内核的任务级响应时间是不确定的，不知道什么时候最高优先级的任务才能拿到 CPU 的控制权，完全取决于应用程序什么时候释放 CPU。不可剥夺型内核允许每个任务运行，直到该任务自愿放弃 CPU 的控制权。中断可以打入运行着的任务。中断服务完成以后将 CPU 控制权还给被中断了的任务。任务级响应时间要大大好于前后系统，但仍是不可知的。

- 可剥夺型内核

当系统响应时间很重要时，要使用可剥夺型内核。最高优先级的任务一旦就绪，总能得到 CPU 的控制权。当一个运行着的任务使一个比它优先级高的任务进入了就绪态，当前任务的 CPU 使用权就被剥夺了，或者说被挂起了，那个高优先级的任务立刻得到了 CPU 的控制权。如果是中断服务子程序使一个高优先级的任务进入就绪态，中断完成时，中断了的任务被挂起，优先级高的那个任务开始运行。使用可剥夺型内核，最高优先级的任务什么时候可以执行，可以得到 CPU 的控制权是可知的。使用可剥夺型内核使得任务级响应时间得以最优化。

使用可剥夺型内核时，应用程序不应直接使用不可重入型函数。调用不可重入型函数时，要满足互斥条件，这一点可以用互斥型信号量来实现。如果调用不可重入型函数时，低优先级的任务 CPU 的使用权被高优先级任务剥夺，不可重入型函数中的数据有可能被破坏。可剥夺型内核总是让就绪态的高优先级的任务先运行，中断服务程序可以抢占 CPU，到中断服务完成时，内核让此时优先级最高的任务运行（不一定是那个被中断了的任务）。任务级系统响应时间得到了最优化，且是可知的。μC/OS-Ⅱ属于可剥夺型内核。

- 可重入性（Reentrancy）

可重入型函数可以被一个以上的任务调用，而不必担心数据的破坏。可重入型函数任何时候都可以被中断，一段时间以后又可以运行，而相应数据不会丢失。可重入型函数或者只使用局部变量，即变量保存在 CPU 寄存器中或堆栈中。如果使用全局变量，则要对全局变量予以保护。

定义为局部变量

调用函数之前关中断，调动后再开中断

用信号量禁止该函数在使用过程中被再次调用

● 时间片轮番调度法

当两个或两个以上任务有同样优先级，内核允许一个任务运行事先确定的一段时间，叫做时间额度（quantum），然后切换给另一个任务。也叫做时间片调度。内核在满足以下条件时，把 CPU 控制权交给下一个任务就绪态的任务：

当前任务已无事可做

当前任务在时间片还没结束时已经完成了。

μC/OS-Ⅱ不支持时间片轮番调度法。应用程序中各任务的优先级必须互不相同。

● 任务优先级

每个任务都有其优先级。任务越重要，赋予的优先级应越高。

静态优先级

应用程序执行过程中诸任务优先级不变，则称之为静态优先级。在静态优先级系统中，诸任务以及它们的时间约束在程序编译时是已知的。

动态优先级

应用程序执行过程中，任务的优先级是可变的，则称之为动态优先级。实时内核应当避免出现优先级反转问题。

● 优先级反转,任务优先级分配

……

● 互斥条件

实现任务间通讯最简便的办法是使用共享数据结构。特别是当所有到任务都在一个单一地址空间下，能使用全程变量、指针、缓冲区、链表、循环缓冲区等，使用共享数据结构通讯就更为容易。虽然共享数据区法简化了任务间的信息交换，但是必须保证每个任务在处理共享数据时的排它性，以避免竞争和数据的破坏。与共享资源打交道时，使之满足互斥条件最一般的方法有：

关中断

使用测试并置位指令

禁止做任务切换

利用信号量

关中断和开中断

处理共享数据时保证互斥，最简便快捷的办法是关中断和开中断。μC/OS-Ⅱ在处理内部变量和数据结构时就是使用的这种手段，即使不是全部，也是绝大部分。实际上μC/OS-Ⅱ提供两个宏调用，允许用户在应用程序的 C 代码中关中断然后再开中断：OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()

● 信号量(Semaphores)

……

● 死锁(或抱死)（Deadlock (or Deadly Embrace)）

死锁也称作抱死，指两个任务无限期地互相等待对方控制着的资源。设任务 T1 正独享资源 R1，任务 T2 在独享资源 T2，而此时 T1 又要独享 R2，T2 也要独享 R1，于是哪个任务都没法继续执行了，发生了死锁。最简单的防止发生死锁的方法是让每个任务都：

先得到全部需要的资源再做下一步的工作

用同样的顺序去申请多个资源

释放资源时使用相反的顺序

内核大多允许用户在申请信号量时定义等待超时，以此化解死锁。当等待时间超过了某一确定值，信号量还是无效状态，就会返回某种形式的出现超时错误的代码，这个出错代码告知该任务，不是得到了资源使用权，而是系统错误。

● 同步

可以利用信号量使某任务与中断服务同步(或者是与另一个任务同步，这两个任务间没有数据交换)。

这个标志表示某一事件的发生(不再是一把用来保证互斥条件的钥匙)。用来实现同步机制的信号量初始化成 0，信号量用于这种类型同步的称作单向同步(unilateral rendezvous)。一个任务做 I/O 操作，然后等信号回应。当 I/O 操作完成，中断服务程序(或另外一个任务)发出信号，该任务得到信号后继续往下执行。

- 事件标志(Event Flags)

当某任务要与多个事件同步时，要使用事件标志。若任务需要与任何事件之一发生同步，可称为独立型同步(即逻辑或关系)。任务也可以与若干事件都发生了同步，称之为关联型(逻辑与关系)。内核支持事件标志，提供事件标志置位、事件标志清零和等待事件标志等服务。事件标志可以是独立型或组合型。

- 任务间的通讯(Intertask Communication)

有时很需要任务间的或中断服务与任务间的通讯。这种信息传递称为任务间的通讯。任务间信息的传递有两个途径：通过全程变量或发消息给另一个任务。

用全程变量时，必须保证每个任务或中断服务程序独享该变量。中断服务中保证独享的唯一办法是关中断。如果两个任务共享某变量，各任务实现独享该变量的办法可以是关中断再开中断，或使用信号量(如前面提到的那样)。请注意，任务只能通过全程变量与中断服务程序通讯，而任务并不知道什么时候全程变量被中断服务程序修改了，除非中断程序以信号量方式向任务发信号或者是该任务以查询方式不断周期性地查询变量的值。要避免这种情况，用户可以考虑使用邮箱或消息队列。

- 消息邮箱(Message Mail boxes)

通过内核服务可以给任务发送消息。典型的消息邮箱也称作交换消息，是用一个指针型变量，通过内核服务，一个任务或一个中断服务程序可以把一则消息(即一个指针)放到邮箱里去。同样，一个或多个任务可以通过内核服务接收这则消息。发送消息的任务和接收消息的任务约定：“该指针指向的内容就是那则消息”。

每个邮箱有相应的正在等待消息的任务列表，要得到消息的任务会因为邮箱是空的而被挂起，且被记录到等待消息的任务表中，直到收到消息。一般地说，内核允许用户定义等待超时，等待消息的时间超过了，仍然没有收到该消息，这任务进入就绪态，并返回出错信息，报告等待超时错误。消息放入邮箱后，或者是把消息传给等待消息的任务表中优先级最高的那个任务(基于优先级)

内核一般提供以下邮箱服务：

邮箱内消息的内容初始化，邮箱里最初可以有，也可以没有消息，将消息放入邮箱(POST)，等待有消息进入邮箱(PEND)，如果邮箱内有消息，就接受这则消息。如果邮箱里没有消息，则任务并不被挂起(ACCEPT)，用返回代码表示调用结果，是收到了消息还是没有收到消息。

消息邮箱也可以当作只取两个值的信号量来用。邮箱里有消息，表示资源可以使用，而空邮箱表示资源已被其它任务占用。

- 时钟节拍(Clock Tick)

时钟节拍是特定的周期性中断。这个中断可以看作是系统心脏的脉动。中断之间的时间间隔取决于不同的应用，一般在 10mS 到 200mS 之间。时钟的节拍式中断使得内核可以将任务延时若干个整数时钟节拍，以及当任务等待事件发生时，提供等待超时的依据。时钟节拍率越快，系统的额外开销就越大。